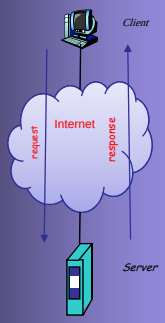


INTERNET PROTOCOLS AND CLIENT-SERVER PROGRAMMING SWE344

Fall Semester 2008-2009 (081)

Module 5.2: C# TCP C/S Programming (Part 2)

Dr. El-Sayed El-Alfy
Computer Science Department
King Fahd University of Petroleum and Minerals
alfy@kfupm.edu.sa

Objectives

- Learn about the **Socket** class
- Learn how to write a TCP server using the **Socket** class
- Learn how to write a TCP client using the **Socket** class
- Learn how to Handle Text in Socket applications
- Learn about some problems that can occur in TCP and how to handle them.

The Socket Class

- The other way to develop C/S applications is using the **Socket** class
  - Powerful class that gives full control
  - Can be used to create C/S applications based on:
    - Various addressing schemes (IP, IPv6, etc)
    - Various Transport Layer protocols (TCP, UDP, etc.)
  - Supports both Synchronous and Asynchronous communications.

- The Constructor of the Socket class has the form:

```
public Socket(AddressFamily af, SocketType st, ProtocolType pt)
```

  - Where each parameter type is an enumeration type in the **System.Net.Sockets** namespace.

The Socket Class …

- **AddressFamily** represents the addressing scheme being used for the communication.
  - Some of the values of this enumeration are: **InterNetwork**, **InterNetworkV6**, **DataLink**, **AppleTalk**, **Ipx**, etc.
  - **InterNetwork** represent IPv4 addressing scheme. All our programs in this course will be centered around this addressing scheme.

- **SocketType** and **ProtocolType** represents the socket type and transport layer protocol for the communication respectively.
  - The following table shows the possible combinations.

Socket Type	Protocol Type	Description
Dgram	Udp	Connection-less
Stream	Tcp	Connection-oriented
Raw	Icmp	Internet Control Message Protocol
Raw	Raw	Plain IP packets

The Socket Class …

- Creating a Socket instance:

```
Socket server = new Socket(AddressFamily.InterNetwork,
                           SocketType.Stream, ProtocolType.Tcp);
```

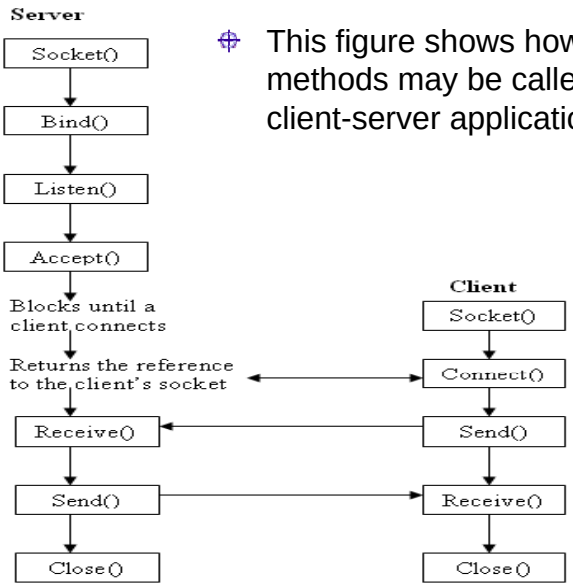
- Some synchronous methods of the Socket class; some of which are used by Server sockets while others are for client.

Method	Description	Used by
void Bind (IPEndPoint ep)	binds a server socket to a local End-Point	Tcp Server
void Listen (int queueSize)	listen for clients; queueSize is the maximum number of clients to enqueue, while waiting for connection	Tcp Server
Socket Accept()	Accepts client's request for connection and returns a reference to its socket	Tcp Server
void Connect (IPEndPoint)	Makes a connection request to a server socket	Tcp Client

The Socket Class …

Method	Description	Used by
int Receive(byte[] data) **int Receive(byte[] data, int size, SocketFlags sf)** **int Receive(byte[] data, int offset, int size, SocketFlags sf)**	reads bytes from a socket	Tcp Server & Client
int Send(byte[]) **int Send(byte[] data, int size, SocketFlags sf)** **int Send(byte[] data, int offset, int size, SocketFlags sf)**	sends bytes to a socket	Tcp Server & Client
void ReceiveFrom(byte[], ref EndPoint)	receives from a client at EndPoint	Udp Client
void SentTo(byte[] ref EndPoint)	sends bytes to a client at EndPoint	Udp Client
void Shutdown(SocketShutdown how)	Disables sends/ receives on socket	All
void Close()	close a socket	All

The Socket Class …



- This figure shows how some of these methods may be called by a TCP client-server application.

Programming a Server Application …

- A version of the Echo server using Socket class:

Server Side

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.Text;
5.  class SimpleTcpSocketServer {
6.    public static void Main() {
7.      Socket server = new Socket(AddressFamily.InterNetwork,
8.                    SocketType.Stream, ProtocolType.Tcp);
9.      IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);
10.     server.Bind(localEP);
11.     server.Listen(10);
12.
13.     Console.WriteLine("Waiting for Client...");
14.     Socket client = server.Accept();
15.     IPAddress clientAddress =
16.                 ((IPEndPoint)client.RemoteEndPoint).Address;
17.     Console.WriteLine("Got connection from "+clientAddress);
```

Programming a Server Application …

```
16.    byte[] data=Encoding.ASCII.GetBytes("Welcome to test server");
17.      client.Send(data);

18.      int size = 0;
19.      while(true) {
20.        data = new byte[1024];
21.        size = client.Receive(data);
22.        if (size == 0)
23.             break;

24.        Console.WriteLine(Encoding.ASCII.GetString(data,0, size));
25.        client.Send(data, size, SocketFlags.None);
26.      }
27.      client.Close();
28.      server.Close();
29.    }
30.  }
```


Programming a Server Application …

Notes:

- The simple version of the **Send** method assumes that the bytes array passed to it as argument is full of data.
- If some cells in the array are blanks, you must use the size argument to specify the actual size of the data.
- Also by default, it assumes that data starts from index zero of the array. If this in not the case, you must specify an **offset** value.
- In the **SimpleTcpSocketServer**, two versions of the **Send** methods were used.
  - First, the simple version was used to send welcome message. This was because the array returned by the GetBytes method in the preceding line has no empty cells.
  - In the second case, we have to use the version of **Send()** that takes size because we are not sure whether the array returned by the preceding call to the Receive method is full of data or not.
- Similar notes applies to the **Receive** method. You must capture the actual data size received and use it in processing the data.

Programming a Client Application …

- A version of the Echo client using the Socket class:

Client Side

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.IO;
5.  using System.Text;
6.  class SimpleTcpSocketClient {
7.    public static void Main() {
8.      Socket socket = new Socket(AddressFamily.InterNetwork,
9.                      SocketType.Stream, ProtocolType.Tcp);
10.     IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse
11.                                  ("127.0.0.1"), 9050);
12.     try {
13.       socket.Connect(remoteEP);
14.     }catch (SocketException e) {
15.       Console.WriteLine("Unable to connect to server. ");
16.       Console.WriteLine(e);
17.       return;
18.     }
```


Programming a Client Application …

```
17.     byte[] data = new byte[1024];
18.     int size = socket.Receive(data);
19.     Console.WriteLine(Encoding.ASCII.GetString(data, 0, size));
20.     String input = null;
21.     while (true) {
22.      Console.Write("Enter Message for Server, Enter to Stop: ");
23.      input = Console.ReadLine();
24.      if (input.Length == 0)
25.           break;
26.      socket.Send(Encoding.ASCII.GetBytes(input));
27.      data = new byte[1024];
28.      size = socket.Receive(data);
29.      Console.WriteLine("Echo: "+ Encoding.ASCII.GetString(
30.                                              data, 0, size));
31.     }
32.     Console.WriteLine("Disconnecting from Server..");
33.     socket.Shutdown(SocketShutdown.Both);
34.     socket.Close();
35.   }
36. }
```

Handling Text in Socket Applications

- In the Echo Client-Server example, we saw that text data had to be converted to bytes array before sending using the Send method.
- Similarly, the Receive method receives the data as bytes array and it had to be converted back to text before printing.
- For exchanging text data, it is not necessary to go through this encoding-decoding process.
- The **NetworkStream** class has a constructor that takes a Socket instance as argument. So a NetworkStream instance can be created from using the Socket instance.
- Also the text handling classes, **StreamReader** and **StreamWriter**, have constructors that takes any **Stream** (FileStream, NetworkStream, etc.) as argument. So instances of these classes can be created from a NetworkStream object.
- With instances of StreamReader and StreamWriter, the familiar Read, ReadLine and Write, WriteLine can be used to exchange text data.

Handling Text in Socket Applications …

- The following is a version of the Echo server that uses text handling methods.

```
1.  using System;
2.  using System.Net;
3.  using System.Net.Sockets;
4.  using System.Text;
5.  using System.IO;
6.  class SimpleTcpSocketServer2 {
7.    public static void Main() {
8.      Socket server = new Socket(AddressFamily.InterNetwork,
9.                      SocketType.Stream, ProtocolType.Tcp);
10.     IPEndPoint localEP = new IPEndPoint(IPAddress.Any, 9050);
11.     server.Bind(localEP);
12.     server.Listen(10);
13.     Console.WriteLine("Waiting for Client...");
14.     Socket client = server.Accept();
15.     IPAddress clientAddress =
16.               ((IPEndPoint)client.RemoteEndPoint).Address;
17.     Console.WriteLine("Got connection from "+clientAddress);
```

Handling Text in Socket Applications …

```
16.      NetworkStream stream = new NetworkStream(client);
17.      StreamReader reader = new StreamReader(stream);
18.      StreamWriter writer = new StreamWriter(stream);

19.      writer.WriteLine("Welcome to my test server");
20.      writer.Flush();
21.
22.      string message;
23.      while((message = reader.ReadLine()) != null) {
24.        writer.WriteLine(message);
25.        writer.Flush();
26.      }
27.      client.Close();
28.      server.Close();
29.    }
30.  }
```


Handling Text in Socket Applications …

- The following is a version of the Echo Client that uses text handling methods.

```
1.   using System;
2.   using System.Net;
3.   using System.Net.Sockets;
4.   using System.IO;
5.   using System.Text;
6.   class SimpleTcpSocketClient2 {
7.     public static void Main() {
8.       Socket socket = new Socket(AddressFamily.InterNetwork,
9.                          SocketType.Stream, ProtocolType.Tcp);
10.      IPEndPoint remoteEP = new
11.             IPEndPoint(IPAddress.Parse("127.0.0.1"), 9050);
12.      try {
13.        socket.Connect(remoteEP);
14.      }catch (SocketException e) {
15.        Console.WriteLine("Unable to connect to server. ");
16.        Console.WriteLine(e);
17.        return;
18.      }
```

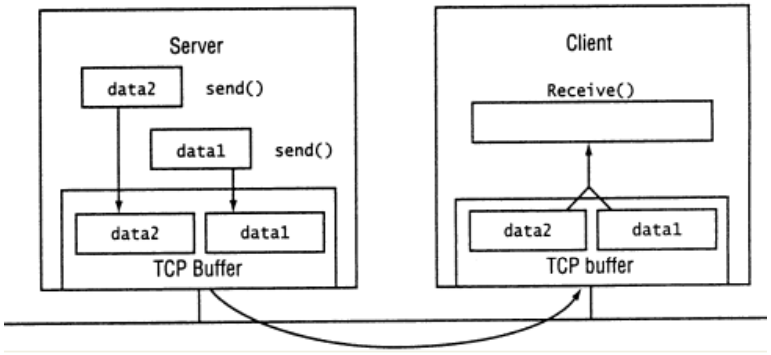
Handling Text in Socket Applications …

```
16.      NetworkStream stream = new NetworkStream(socket);
17.      StreamReader reader = new StreamReader(stream);
18.      StreamWriter writer = new StreamWriter(stream);
19.      Console.WriteLine(reader.ReadLine());
20.      String input = null;
21.      while (true) {
22.        Console.Write("Enter Message for Server, Enter to Stop:");
23.        input = Console.ReadLine();
24.        if (input.Length == 0)
25.             break;
26.        writer.WriteLine(input);
27.        writer.Flush();
28.        Console.WriteLine("Echo: "+ reader.ReadLine());
29.      }
30.      Console.WriteLine("Disconnecting from Server..");
31.      socket.Shutdown(SocketShutdown.Both);
32.      socket.Close();
33.    }
34.  }
```


Problems in TCP Communication

- Due to stream nature of data exchange in TCP, some problems may be encountered:
  - Too Small Buffer Size
    - In real world, you may not know the size of the data. So what happens if more data arrives than the buffer size?
    - In our simple examples, a byte array of size 1024 was used as the buffer size for the **Send** and **Receive** method calls.
    - This worked fine because the program was running in a controlled environment, i.e. both server and client know that the message size will not be more than this size.
  - Message Boundary Problem
    - This problem is due to its connection-oriented nature, messages are considered to form a continuous stream of bytes.
      - TCP uses internal buffers to store messages until they are Received/Sent by applications
      - This implies that TCP does not respect message boundaries, i.e. there is no one-to-one correspondence between number/size of individual messages sent and the number/size of individual messages received.
    - TCP pair (on the Sever and Client side) will ensure that no data is lost.
    - The problem is with the applications. How will they know how many times they need to read before they collect the whole message?

Problems in TCP Communication …



Problems in TCP Communication …

Solutions (1):

- For Text messages only, a solution is to use **ReadLine** and **WriteLine** methods of the **StreamReader** and **StreamWriter** classes respectively.
  - **WriteLine** at the sender will insert end-of-line markers in the message; thus creating a boundary.
  - **ReadLine** at the receiver will read one line at a time until there is no more lines to read.
- Notice that the problem of too small buffer does not even arise in this case, since the **ReadLine** and **WriteLine** methods will take care of this.

Problems in TCP Communication …

Solutions (2):

- Send the **size** of the message first, before sending the message.
  - This is applicable for any type of data where **Send** and **Receive** methods are used for sending and receiving.
- Since the receiver knows the total size of the data, it will read the data in a loop, each time taking note of the actual size returned by the **Receive** method and updating the amount so far read until the entire size is read.
- A loop similar to the following is used:

```
1.       int total = int.Parse(reader.ReadLine());
2.       byte [] buffer = new byte[1024];
3.       int recv = 0;  int sofar = 0;
4.
5.       while (sofar < total) {
6.         recv = s.Receive(buffer);
7.         process(buffer, recv);
8.         sofar += recv;
9.       }
```


Resources

- MSDN Library
  - http://msdn.microsoft.com/en-us/default.aspx
- Books
  - Richard Blum, C# Network Programming. Sybex 2002.
- Lecture notes of previous offerings of SWE344 and ICS343
- Some other web sites and books; check the course website at
  - http://faculty.kfupm.edu.sa/ics/alfy/files/teaching/swe344/index.htm